

Lab 04: Exception Handling and Algorithms

This lab introduces error handling in Python. In addition, you will use the `os` module to work with external files and paths. We then move on to explore a number of algorithms.

Coding "patterns" include recursive functions and nested functions.

The case study is a spell checker built in Python, which can use two different search algorithms. This will be extended next week by adding a third algorithm and then comparing the efficiency of the different algorithms.

1. External Data – User Input Exceptions

Run the `add_2_ints.py` script you created for Exercise 5 Lab 3 (last week), but this time enter `'knight'` as the first number.

Q: What happens?

Q: What specific type of Exception is raised?

You should get an Exception, and a Traceback which shows exactly where the run-time error occurred, and the error itself (last line) and the type of error; in this case a ***ValueError***.

```
Please enter a number: knight
Please enter a 2nd number: 4
Traceback (most recent call last):
  File "C:\Users\Petra\Dropbox\CSN08114 Python\add_2_ints.py", line 15,
in <module>
    total = int(reply1)+int(reply2)
ValueError: invalid literal for int() with base 10: 'knight'
```

Trapping the error

We can use the "try/except" construct to check for a `ValueError` and carry out a different action if it has occurred. Typically, the action is to print a tailored error message for the user.

This could be done like the code shown below.

Change your script as shown below and then test it:

- If given two numbers it should run exactly as before;

- otherwise it should print the error message.

```
# Module: add_2_ints.py
# Desc: Get two numbers from user and add them together
# Modified: Oct 2017

try:
    reply1 = input('Please enter a number: ')
    reply2 = input('Please enter a 2nd number: ')
    total = int(reply1)+int(reply2)
    print (f'Sum = {total}')
except ValueError:
    print ('Please supply integers')
```

Why did we convert the input to int? Let's see what happens if we don't.

Change a copy of the script (give it a different name!) so that reply1 and reply2 are NOT converted to integers before adding up.

Q: Run the modified script with Hello and World as inputs. What happens? Why?

Q: Run the modified script with 6 and 7 as inputs. What happens? Why?

Q: What happens if you give a number and a string as inputs?

Q: You could use eval() instead of int(), but why is int() better in this context?

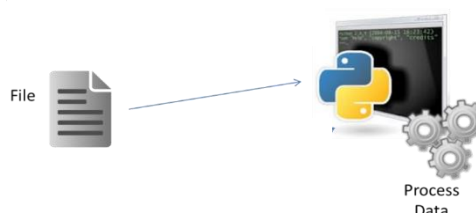
Read <https://stackoverflow.com/questions/45881547/python-what-is-the-difference-between-eval-and-int> and briefly explain in your own words.

2. Optional Challenges ***Optional***

- Change the add_2_ints.py script so it correctly adds up the numbers if numbers are given, concatenates strings if two strings are given, and prints an error message if a number and a string are given?
- Change the error handling so that if a number and string are given, not only is an error message printed, but also the script asks for input again. (Hint: use a while loop)

3. The os module – Input from Files (ctd)

In the previous lab, lab 3, we read external data from a file. We used the open() and read() or related methods. In our examples, we assumed the file we were reading from was in the same directory.



But what if the file is in a different directory? We could of course specify the filename using the whole path, e.g. "C:\Users\Petra\CSN08114\dictionary.txt".

BUT...Python itself is operating system independent - using a path like the above would only work in Windows, and would introduce a dependency on the operating system.

To solve this problem, we can use the Python module **os**. The functions in this module let us interact with the underlying File System and keep the code operating system independent as much as possible.

Import the **os** module, and use <CTRL-SPACE> to scroll through the available functions.

```
>>> import os
>>> os.fdopen
execvpe
extsep
fdopen
fstat
fsync
getcwd
getcwdu
getenv
```

Q: Which function might help with changing directory on the underlying file system?

Create a new directory in **C:\temp**, and copy the **dictionary_variants.txt** file there (available in moodle - you should have downloaded it for lab 3 already).

Check the current working directory using the **os.getcwd()** function.

```
>>> import os
>>> os.getcwd()
'C:\\Program Files\\Python36'
```

List the files in the current working directory with **os.listdir()**:

```
>>> os.listdir()
['DLLs', 'Doc', 'include', 'Lib', 'libs', 'LICENSE.txt', 'NEWS.txt', 'python.exe', 'python.pdb', 'python3.dll', 'python36.dll', 'python36.pdb', 'python36_d.dll', 'python36_d.pdb', 'python3_d.dll', 'pythonw.exe', 'pythonw.pdb', 'pythonw_d.exe', 'pythonw_d.pdb', 'python_d.exe', 'python_d.pdb', 'Scripts', 'tcl', 'timing_comprehensions.py', 'Tools', 'vcruntime140.dll']
```

Use the **os.chdir()** function to change to the directory where the dictionary.txt file is now.

```
>>> os.chdir('C:\\temp')
```

Check the dir has been changed using the previous command (CTRL+P).

```
>>> os.getcwd()
'C:\\temp'
```

List the files in the **c:\temp** directory to check dictionary.txt is there.

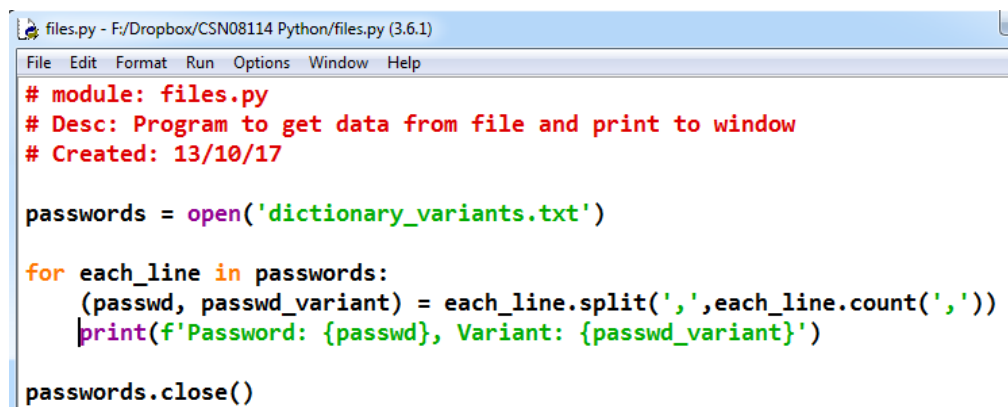
Q: Which type of object is returned by `os.listdir()`?

We will use this more later.

4. More Python Exception Handling: try and except

Run-time errors should ideally be anticipated and handled in the code, thus preventing our script crashing.

For this exercise, we will use the same code as for exercise 9 in lab 3, which is shown below. Copy the code and save it as a Python script called `files.py`.



```
files.py - F:/Dropbox/CSN08114 Python/files.py (3.6.1)
File Edit Format Run Options Window Help
# module: files.py
# Desc: Program to get data from file and print to window
# Created: 13/10/17

passwords = open('dictionary_variants.txt')

for each_line in passwords:
    (passwd, passwd_variant) = each_line.split(',')
    print(f'Password: {passwd}, Variant: {passwd_variant}')

passwords.close()
```

Run the `files.py` script and you should get the same Exception encountered in exercise 9 lab 3.

Q: In the listing above, circle the code which may cause the exception, and also any dependent code.

The code which caused the error and the code directly after it should be grouped together with a `try` statement, and an `except` statement should be created with code to handle the exception.

In this case, print that the line cannot be handled by the Tuple.

```
# module: files.py
# Desc: Program to get data from file and print to window
# Created: 13/10/17

passwords = open('dictionary_variants.txt')

for each_line in passwords:
    try:
        (passwd, passwd_variant) = each_line.split(',')
        print(f'Password: {passwd}, Variant: {passwd_variant}')
    except:
        print('line does not conform to expected format')

passwords.close()
```

Q: Did this stop the crash and Traceback error?

```
        Password: qwerty, Variant: qwerty1
passwords = open('dictionary_variants.txt')

for each_line in passwords:
    try:
        (passwd, passwd_variant) = each_line.split(',')
        print(f'Password: {passwd}, Variant: {passwd_variant}')
    except:
        pass

passwords.close()
```

We could also simply ignore the line, using the dummy statement `pass`. Try this for yourself by saving a copy of `files.py` as `files1.py` and changing it to:

You should find that any errors with the `split` command are ignored, and the rest of the file will be happily processed.

5. File Handling Exceptions

Typically, Exception handling is used to cope with **unexpected runtime errors**. If you are working with external files, a typical problem is that the file has a different name or is in a different location than expected. Then, if we try to open the file, the program will crash, and an Exception will be raised.

Rename the file **`dictionary_variants.txt`** to **`dictionary1.txt`**. Run `files.py` again, without modifying it.

Q: What happens?

Q: What type of Exception is raised now?

An Exception of the type `FileNotFoundError` will be raised, such as the following.

```
Traceback (most recent call last):
  File "F:/Dropbox/CSN08114 Python/files.py", line 5, in <module>
    passwords = open('dictionary_variants.txt')
FileNotFoundError: [Errno 2] No such file or directory: 'dictionary_variants.txt'
```

In the `files.py` module, add a second layer of Exception handling to protect the `file.open()` statement and dependent code, and giving the user a message **'Error with dictionary_variants.txt file'** if an error is raised.

Note that it is good practice to catch exceptions as close to the point they're thrown as possible - it makes the code easier to read and avoids catching unrelated exceptions of the same type.

To allow for this, use the structure:

```
try:
except:
else:
```

In this case the else: block will run when no error is raised, i.e. when the file is successfully opened.

Run and test your code, so the Traceback is not displayed to the user.

Q: Does your code run and only the error message is shown to the user?

The code should now handle the File Exception:

Error with dictionary_variants.txt file

Your code should be similar to the following:

```
try:
    passwords = open('dictionary_variants.txt')
except:
    print('Error with dictionary_variants.txt file')
else:
    for each_line in passwords:
        try:
            (passwd, passwd_variant) = each_line.split(',')
            print(f'Password: {passwd}, Variant: {passwd_variant}')
        except:
            print('line does not conform to expected format')

    passwords.close()
```

Q: Why is the close() method called inside the else block?

6. Using specific Exception Types

Now let's check specifically for a **FileNotFoundError** Exception type.

To check for the different types of exception, after the **except** statement use CTRL+SPACE. Scroll down and select the **FileNotFoundError** type.

Change the exception to trigger specifically only with a **FileNotFoundError**, and use the error properties in the printed error message:

```
except FileNotFoundError as err:
    print(f'{type(err)}: {err}')
```

Q: Does your code run and show the correct error message to the user?

7. Reading dictionary_variants.txt into a list

In exercise 9 Lab 3, we found that a Tuple is a good collection object to use if the data being read are in a fixed format. However, the error we trapped in Exercise 4 above occurs if the data format varies. Our solution in Exercise 4 allowed the rest of the file to be read but ignored the line in password_variants.txt that has two variants. So how can we change our script so that each line can contain several variants?

Let's try using a **List** object to collect the data from the file instead of a tuple:

```
try:
    passwords = open('dictionary_variants.txt')
except FileNotFoundError as err:
    print(f'{type(err)}: {err}')
else:
    passwds = []
    for each_line in passwords:
        try:
            passwds.append(each_line.split(',', each_line.count(',')))
            #print(f'Password: {passwd}, Variant: {passwd_variant}')
        except:
            pass

    passwords.close()
    print(passwds)
```

Before you run this, make sure you have put the dictionary_variants.txt file back where it should be, so the file can be opened.

Q: What does the `passwds []` list contain?

Q: What does the `'\n'` in the last item on each line mean?

8. Chomping Strings

The `'\n'` is the newline character. We can remove this from the end of the line before we split it using the `string.strip()` method, which also removes any whitespace. This is sometimes called *chomping* a string.

In the Python shell, try the following:

```
>>> chompable_str = 'Pesky newline char str\n'
```

Now print the string out.

Q: What do you get?

You get an extra newline, as the string contains a newline char and the print() function also adds a newline char by default.

Chomp the string and print it out with:

```
>>> chomp='pesky newline char\n'
>>> chomp
'pesky newline char\n'
>>> print(chomp)
pesky newline char

>>> print(chomp.strip())
pesky newline char
```

Back in `files.py`, create a line of code to chomp the `each_line` string before appending it to the list.

Q: What is the code?

Test the code.

Q: Are the newline characters now removed?

Your changed code block and testing output should look similar to:

```
for each_line in passwords:
    try:
        passwds.append(each_line.strip().split(',',each_line.count(',')))
        #print(f'Password: {passwd}, Variant: {passwd_variant}')
    except:
        pass
```

```
>>>
[['qwerty', ' qwerty1'], ['password', ' password1'], ['default', ' default1'], [
'123', ' 12345', ' 123456']]
>>> |
```

Q: Where is the strip() method called in the example code above?

Q: Why is this preferable to adding the separate line:

`each_line = each_line.strip()`
before the `passwds.append()` line?

9. Case study part 1: testing search algorithms

In the lecture, we discussed two different search algorithms, linear and binary search.

Let's start by exploring these more fully.

Start by creating or downloading the two files below.



Script starting points:

SearchLinear.py and SearchBinary.py (in Moodle)

```
# SearchLinear.py
# linear search for spell check

def linS(target, words):
    '''linear search for target in words. words need not be sorted'''
    for s in words:
        if s==target:
            return True
    return False

#Get a list of words
words = [s.strip("\n").lower() for s in open("words.txt")]
# tests
for w in ["accomodation","buchanan","macintosh","zebede"]:
    if not linS(w, words):
        print (w)
```

```
# SearchBinary.py
# Binary search for spell check

def binS(lo,hi,target):
    '''binary search for target in words.
       words must be declared elsewhere and a sorted list.'''
    if (lo>=hi):
        return False
    mid = (lo+hi) // 2
    piv = words[mid]
    if piv==target:
        return True
    if piv<target:
        return binS(mid+1,hi,target)
    return binS(lo,mid,target)

#Get a list of words
words = [s.strip("\n").lower() for s in open("words.txt")]
words.sort() # sort the list
# tests
for w in ["accomodation","buchanan","macintosh","zebede"]:
    if not binS(0,len(words),w):
        print (w)
```

Q: What do the scripts need in order to run without error?

Download the file words.txt from moodle and save it in the same directory as the scripts.

Now you're ready to try out the scripts.

Run each of the scripts in turn.

Q: Do both scripts give the same results?

Download another wordlist from <https://raw.githubusercontent.com/first20hours/google-10000-english/master/20k.txt>. This is a smaller "dictionary", it contains approx. 20,000 words, while words.txt contains around 100,000.

Rerun both scripts using this word list instead of words.txt.

Q: Are the results the same as with the larger wordlist?

Q: Do both scripts still give the same results?

Q: Comment out the line that sorts the wordlist. Rerun both scripts. Do the results change? What exactly is different?

To find out why the results change, open both of the word list files in a text editor (I recommend Notepad++, as Windows Notepad may struggle with large files like these).

Q: Apart from the size of the text files, how is their structure different?

Q: What might be the reason for 20k.txt being the structure it is?

Q: Why does this cause a problem for binary search?

10. Case study part 2: Pre-processing a novel for spell-checking

Project Gutenberg (<http://www.gutenberg.org>) has digitised many classic works of literature that are copyright free. We will use Frankenstein by Mary Wollstonecraft Shelley.

Download the complete text of this novel as a text file from <http://www.gutenberg.org/files/84/84-0.txt>. Rename your file to **Frankenstein.txt**.

Open the file in a text editor (Notepad++ recommended) and inspect it.

Q: What is the encoding of this text file?

Q: How many words are on each line?

Q: Does the file contain characters that should be removed before spell checking?

As the novel contains more than one word per line, as well as punctuation and other formatting characters, we need to pre-process it. The result of this pre-processing should be a text file **cleaned.txt** which

- contains one word per line,
- all in lower case, and
- without any punctuation or special characters.
- In addition, it should not contain any "words" that consist of just one character
- nor should it contain any empty lines.

Your task is to write a script to read Frankenstein.txt, pre-process it, and write the resulting "cleaned" text to a new file **clean.txt**.

The first few lines of clean.txt should look like this → →:

Use the lecture slides and material from earlier lectures and labs to help you. Remember that the Python docs and google are also a good source for help.

When writing to a file, you may need to add in line feeds (\n). If you don't do this, you may find that the new file contains everything rolled together into a single, extremely long, word.

```
project
gutenberg
frankenstein
by
mary
wollstonecraft
godwin
shelley
this
ebook
is
for
the
use
of
anyone
anywhere
at
no
cost
and
with
almost
no
restrictions
```

11. Case study part 3: Putting it all together

Create a new module, **spellcheck.py**.

a) Copy the two search functions into this module. (In this case, we want all functions to be in one place).

b) Write a function `main()`.

`Main()` needs to do the following:

- Read in `cleaned.txt`
- Read in a wordlist to spell check against
- Call each of the two spell checking functions in turn.
- For each of the two function calls, print how many words are not in the dictionary
- Check whether both functions give the same results, and report (print) the outcome.

c) Add the standard boiler plate.

This time, you need to structure the code yourself. Remember doc strings and comments to help others understand your code.

Get help in the lab and show your code to a tutor once you are finished, for feedback.

Test your program, using three different word lists (two of which you have used already) :

- 10,000 words <http://www.mit.edu/~ecprice/wordlist.10000>
- 20,000 words <https://raw.githubusercontent.com/first20hours/google-10000-english/master/20k.txt>
- 100,000 words <http://www.soc.napier.ac.uk/~40009856/words.txt> (this is a copy of Ubuntu's built-in file `british-english`).

Compare the results with others.

You will need this as a starting point for continuing the case study next week.